Predicting CS:GO round winner using non-probabilistic supervised classification

Iñigo Maiza Razkin

Universidad Politécnica de Madrid (UPM)

Code of the project available at:
https://github.com/imaiza

Abstract- In this project, I used a CS:GO dataset, in order to predict the round winner using 5 different supervised classification algorithms (k-NN, NN, SVM, Decision trees and RIPPER), obtaining an accuracy up to 82%. However, this project will be focused on understanding the different algorithms, how do they work and how do they perform on different subsets of our dataset; rather than tuning them to perform better.

I. INTRODUCTION

Counter Strike Global Offensive(CS:GO) is a popular first person shooter (FPS) videogame developed by "Valve Corporation". The game was released in 2012 and has increased his popularity and playerbase since then, becoming the 3rd most played online videogame in 2019 [1].

The game pits two teams (5 players each) against each other: the Terrorists(T) and the Counter-Terrorists(CT). Both sides are tasked with eliminating the other while also completing separate objectives. The Terrorists, must plant the bomb to win a round, while the Counter-Terrorists must prevent the bomb from being planted, or defuse the bomb (if it was planted). Another way of winning a round consists on killing all 5 members of the other team. The first team winning 16 rounds wins the game. The game also has a complex economy system, that allows players to buy different guns and utilities (such as armor or grenades). Teams earn money by winning rounds and they spend it on new armament.

II. PROBLEM DESCRIPTION

In this project, I will try to predict the round winner in a CS:GO match using non-probabilistic supervised classifiers. More specifically, 5 classifiers will be used: k-nearest neighbors, rule induction (RIPPER), support vector machines, neural networks and classification trees.

The dataset that I will be using during the study is taken from Kaggle [2]. The dataset consists of ~700 games from high level tournament play in 2019 and 2020. Warmup rounds and restarts have been filtered, and for the remaining live rounds a round snapshot have been recorded every 20 seconds until the round is decided. The total number of snapshots is 122411 (observations). The number of variables is 20: *time_left, ct_score, round, bomb_planted, ct_health,t_health,ct_armor,t_armor,ct_money,t_money,ct_helmets,t_helmets,ct_defuse_kits,ct_players_alive,t_players_alive,ct_weapon_ak47,t_weapon_ak47,ct_weapon_awp, t_weapon_awp and round_winner.*
The money of the Terrorist team is labeled as "t_money" and same for "ct_money". "bomb_planted" is a Boolean variable (variable telling if bomb has been planted or not). The variables(*ct_weapon_ak47,t_weapon_ak47,ct_weapon_awp,t_weapon_awp)* provide information about the most important and decisive weapons (AK47 and AWP) both team have: they count +1 for every team member using the weapon. This variables are ranged from 0 (no one on the team has it) to 5 (all members have it). The *'round_winner'* variable will be the target of our classification problem, and it takes two values: T (Terrorist team wins) and CT (Counter-Terrorist team wins).

First of all, I did a preliminary study of our dataset, in order to figure out if a binary classifier could perform efficiently. I plotted the features against each other (except Boolean or discrete valued ones) and I obtained figure 1 (*Appendix*). We can easily see that using supervised learning classification algorithms seems like a good idea, given that on some of the plots blue/orange (T/CT) points can be separated. We will train our algorithms so they separate the points in the best way possible.

Furthermore, I did another preliminary verification in the dataset, in order to check if the different variables are correlated. The results are plotted in *Figure 2*. Here we see that the features are not highly correlated to each other, except from little clusters such as the one formed by 'round' and 'ct_score', which is logical given that 't_score'+'ct_score' = 'round'.

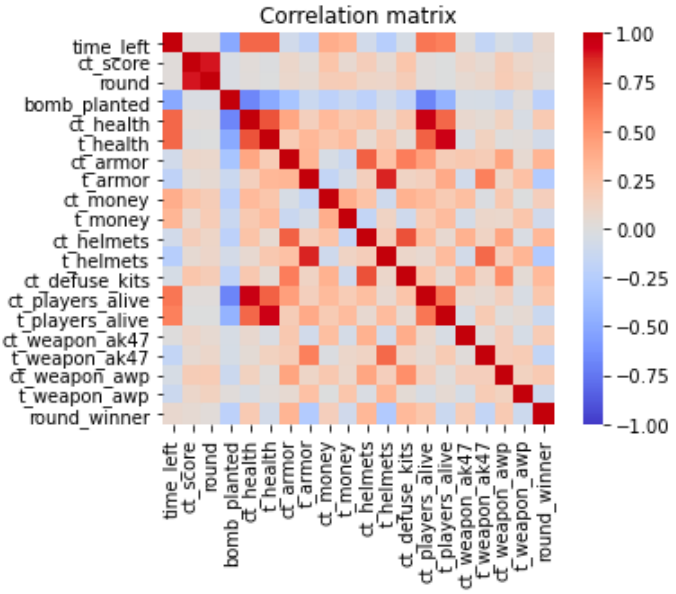

***Figure 2**: Correlation matrix of the variables.*

In short, I did some testing to the dataset concluding that we will work with a proper dataset.

III. METHODOLOGY

The whole project has been developed using Pyhton, and the majority of the classifiers where taken from the sklearn library [3]. The rule induction based RIPPER algorithm was taken from the Wittgenstein library [4]. The training and testing set are divided 80-20% randomly using 'train_test_split' function from sklearn, and the 'accuracy' that we will use as our score metric during this study is the percentile of the correct predictions of the classifier in the test set (a perfect classifier would always have accuracy=1).

This work is divided in 4 analysis: using the whole dataset, with a univariate filter feature subset selection; with a multivariate filter feature subset selection and with a multivariate wrapper feature subset selection. In each analysis, the working methodology will be the one described above. In the first analysis (all variables), the results will be analyzed individually for each algorithm, in an effort to understand and interpret the models at the beginning of the study, before analyzing how do different subset affect them.

Moreover, it is worth pointing out that the goal of this project is not finding, training and tuning the parameters of a given classifier to have the highest accuracy possible. This project is focused on understanding the different algorithms, how do they work and how do they perform on different subsets of our dataset.

IV. RESULTS AND DISCUSSION

IV.I) ALL VARIABLES

In this first approach, we will use the whole dataset (20 variables) to train our algorithms. After training all of them, this is their accuracy predicting the test set:

```
Accuracy for k-NN: 0.8135628805374764
Accuracy for RIPPER: 0.6938904052068025
Accuracy for Neural Networks: 0.7270627755616208
Accuracy for Decision Trees: 0.8290993071593533
Accuracy for Svm: 0.6546294352298971
```

***Figure 3**: Accuracy for all the classifiers.*

To visualize this better, we plot the ROC curve of the algorithms (given RIPPER algorithms' nature, I haven't been able to plot his curve):

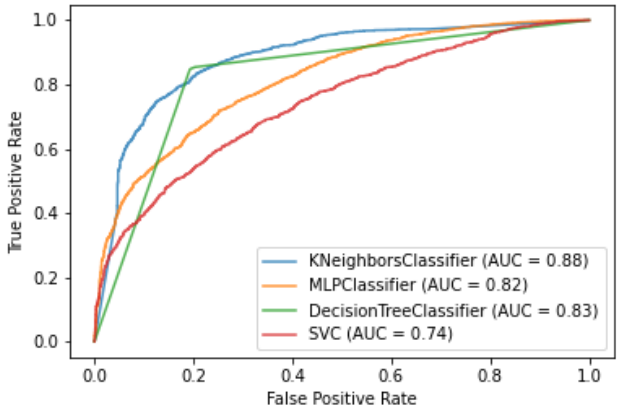

***Figure 4**: ROC curve with all variables.*

First of all, it looks like there are two classifiers that perform better than the others (k-NN and Decision Trees) with 0.81-0.82 accuracy. Neural networks, RIPPER and especially support vector machines don't seem to perform as good with 0.73, 0,69 and 0.65 accuracy respectively.

In order to understand this results and the algorithms better, we will analyze them separately:

A) Support Vector Machine (SVM)

The used SVM for the study was the sklearn.svm.SVC from the sklearn library. As we saw in the *Figure 1*, our data points are heavily overlapped so building a SVM using a linear model is probably not a good idea. I tried doing so, and as expected, training a linear SVM was impossible. Even trying to implement a "soft-margin SVM" was impossible, given that there are too many outliers to

train a good model. The only approach I had left was using a non-linear kernel, so I used the Gaussian RBF kernel. The accuracy of this model is the one presented before: **0.654**. If we plot de confusion matrix:

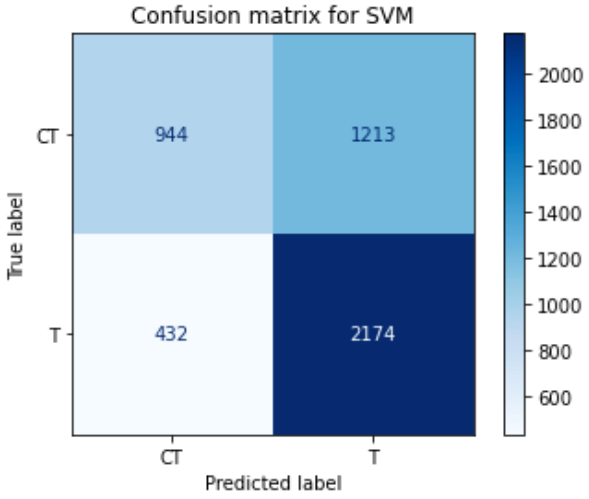

***Figure 5**: Confusion matrix for SVM(kernel=rbf).*

The accuracy of the model is pretty bad. In order to understand what is our model doing, we plot the decision regions in *Figure 6*. We choose two features based on *Figure 1*: we select a plot where a classification task should be easy to execute (t_money/t_armor).

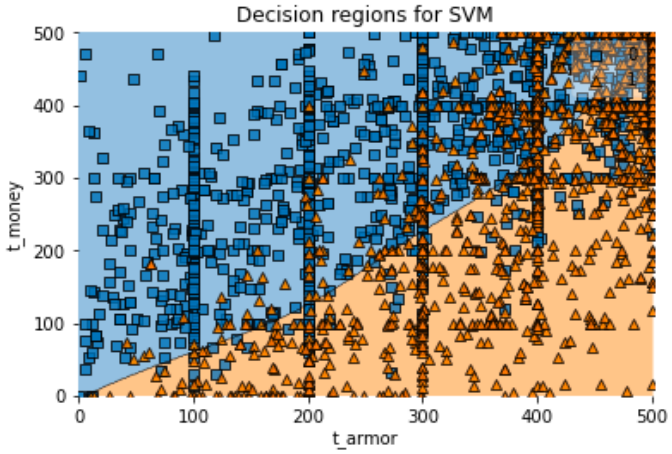

Figure 6**: Decision regions, taking 20% of the whole dataset to visualize the regions better.**Orange=CT,Blue=T.

We see that our model is defining a curved line and all the points above it are classified as T, and vice versa. As stated before, we have a lot of overlapping points in our problem, so the model will classify a lot of points inappropriately. That explains our low accuracy using this model. If we want to improve the accuracy of our predictions, we need to use a more complex model.

B) K -Nearest Neighbor

In this case I used the sklearn.KNeighborsClassifier from the library sklearn with default values (neighbors = 5). As we saw, the accuracy for this model is one of the highest: **0.814**. The confusion matrix for this case:

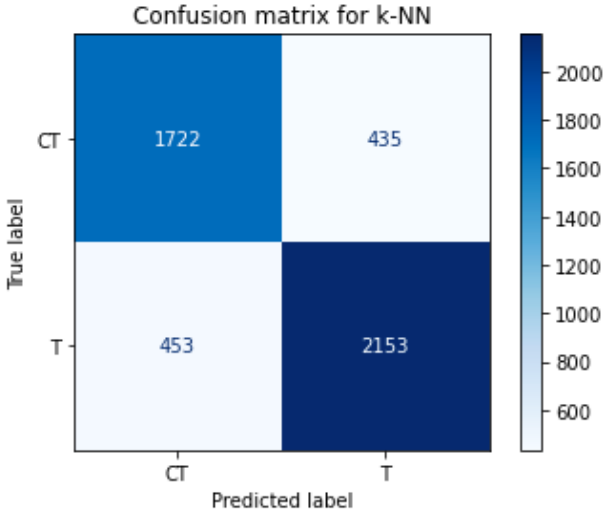

***Figure 7**: Confusion matrix for k-NN.*

Here we see that the result is much better than in the previous case. We plot the decision regions again in the *Figure 8*:

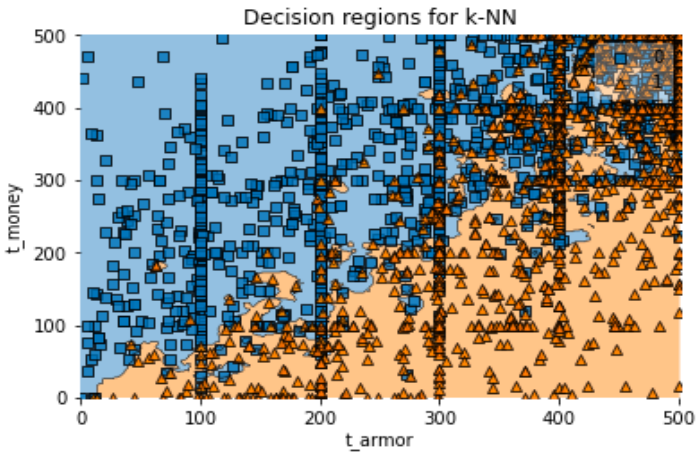

Figure 8**: Decision regions, taking 20% of the whole dataset to visualize the regions better.**Orange=CT,Blue=T.

We see how the k-NN algorithm classifies correctly some of the points that are on the other side of the line defined by the SVM model. In fact, we see how the k-NN model is able to learn complex decision boundaries surprisingly well. We see how k-NN finds clusters of CT/T data surrounded by T/CT data. This is the reason why k-NN outperforms SVM and has a high accuracy score.

C) Decision Tree

The used tree was sklearn.tree.DecisionTreeClassifier from sklearn library (default values, criterion=gini). In this case the accuracy was even higher than for the k-NN model**, 0.829.** Plotting the confusion matrix we get:

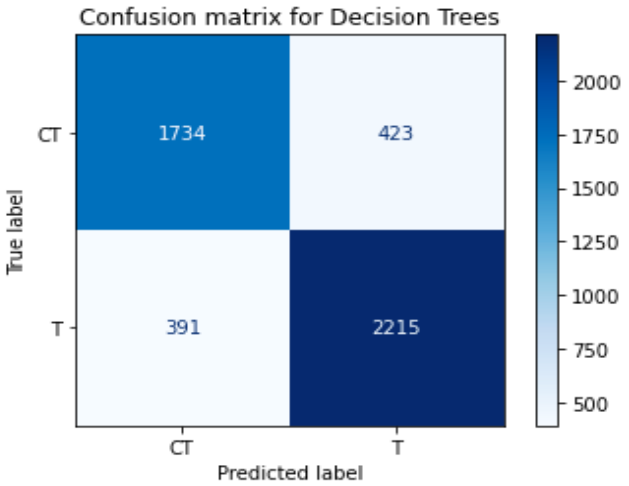

***Figure 9**: Confusion matrix for Decision Trees.*

This is the best model we have seen so far, so we plot in *Figure 10* the decision regions of this classifier to understand how is working:

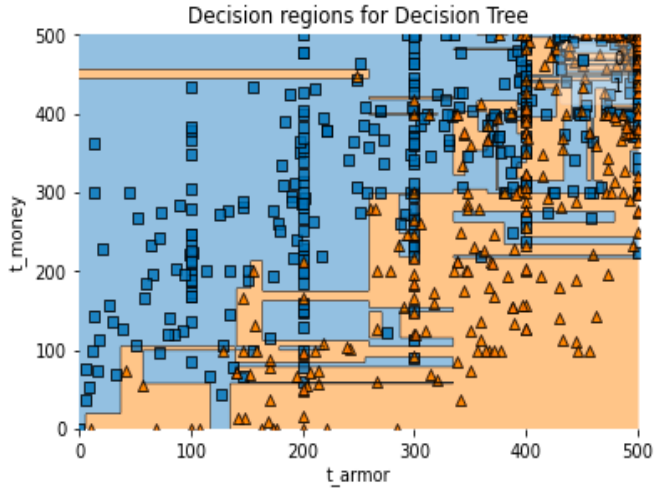

Figure 10**: Decision regions, taking 5% of the whole dataset to visualize the regions better.**Orange=CT,Blue=T.

In this case, we see how the regions are squared, where the limits of this regions are defined by the complex decision rules of our trained Decision Tree model. We clearly see how this approach defines regions in a really efficient way. If we visualize our tree (the top of it) we can see the rules that separate the regions (*Figure 11*, see *Appendix).*

In *Figure 11*, we see the top of the tree, where both teams armor ('t_armor' and 'ct_armor') appear like the principal filters. In fact, 'ct_armour' is chosen as de root node of the tree.

In short, decision trees are a really good model for our problem, given that we have a lot of overlapping points, and the complex decision rules of the model are really efficient on separating and classifying those points.

D) Rule Induction (RIPPER)

The chosen rule induction algorithm for this work was the RIPPER algorithm, taken from the Wittgenstein library. The accuracy of this model is not as good as the previous' one, **0,694.** This algorithm defines simple rules on the training, as we can see in *Figure 12*.

```
[bomb_planted=False ^ ct_helmets=3.0 ^ ct_score=2.0-4.0 ^ t_armor=472.0-495.0] V
[bomb_planted=False ^ ct_helmets=3.0 ^ t_players_alive=2.0 ^ t_health=0.0-170.0 ^ ct_score=6.0-8.0] V
[bomb_planted=False ^ t_helmets=0.0 ^ t_armor=0.0-0.0 ^ ct_armor=200.0-300.0 ^ time_left=169.95-174.95] V
[bomb_planted=False ^ ct_helmets=3.0 ^ t_helmets=1.0 ^ t_weapon_ak47=1.0 ^ ct_defuse_kits=2.0] V
[bomb_planted=False ^ ct_defuse_kits=3.0 ^ t_armor=164.0-231.0] V
[ct_players_alive=5.0 ^ ct_helmets=5.0 ^ ct_score=12.0-15.0 ^ t_weapon_awp=1.0 ^ ct_weapon_ak47=2.0]]
```

***Figure 12**: Some rules of our RIPPER model.*

We can see how this model builds simple rules to use them later to make predictions. This rules, in contrast of the rules of the decision tree, are not hierarchical. Therefore, this model is unable to create very complex rules. This is the reason why the accuracy of the model is not as high as in decision trees; because we have a complex problem with a lot of features.

E) Neural Networks (NN)

I used the sklearn library multilayer perceptron with default values: 'sklearn.neural_network.MLPClassifier' . The accuracy for this model was **0.727.** The confusion matrix in this case:

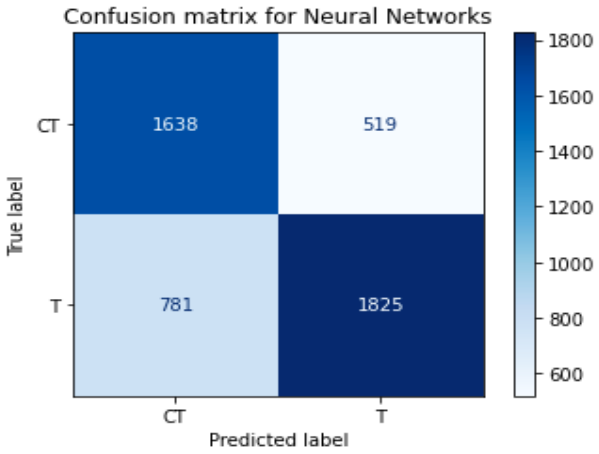

***Figure 13**: Confusion matrix for NN*

We see that the accuracy of this model is not that high, even though neural networks are supposed to be a really powerful tool. We plot the decision regions to understand what is happening:

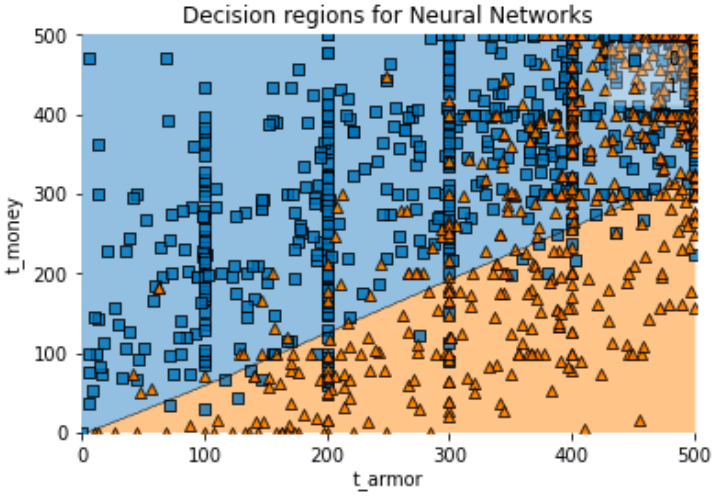

Figure 14**: Decision regions, taking 10% of the whole dataset to visualize the regions better.**Orange=CT,Blue=T.

Whit the help of this figure we realize that we have the same problem that we had with the SVM. Our model generates a straight line, and we already demonstrated that a linear classifier is an awful idea for our problem. This is the reason why this classifier gets a unexpectedly bad accuracy.

Neural networks are a really powerful tool, and they have proved to be able to solve really complex problems. In this study I used the simplest classifier, with default values. If we tune the parameters of the NN properly, it should be able to solve this problem with a significantly better accuracy, but that is not the point of this project.

IV.II) UNIVARIATE SUBSET SELECTION

In this second approach, we will use the sklearn feature sklearn.feature_selection.SelectKBest to filter our dataset using the sklearn.feature_selection.chi2 function. We will create a subset of 11 variables, reducing the number of variables from 20 to 11 using the chi-squared method.

After applying this function to the dataset, we end up with these 11 variables: 'time_left', 'ct_health', 't_health', 'ct_armor', 't_armor', 'ct_money', 't_money', 'ct_helmets', 't_helmets', 'ct_defuse_kits', 'round_winner'.

Once we defined our new subset, we train our 5 models and we test them. The accuracies are shown in *Figure 15*.

```
Accuracy for k-NN: 0.8099937014486668
Accuracy for RIPPER: 0.6592483728742389
Accuracy for Neural Networks: 0.7247533067394499
Accuracy for Decision Trees: 0.811043460004199
Accuracy for Svm: 0.6544194835187906
```

***Figure 15**: Accuracy for all the classifiers.*

We also plot the ROC curve so we visualize the result better:

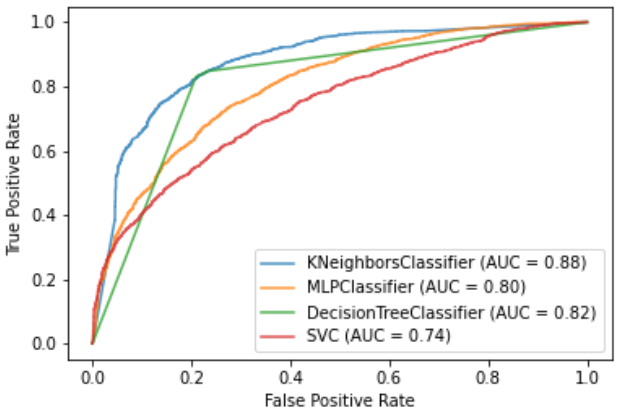

***Figure 16**: ROC curve with univ. subset selection*

We see that the accuracy of all methods descends slightly, but the time needed to train all algorithms drops significantly. After applying this univariate subset selection we lose half of the features, but the models are still able to perform in a similar way, and they are much faster.

This method will be really useful when you have big datasets and you don't mind losing some accuracy in exchange of speed.

IV.III) MULTIVARIATE SUBSET SELECTION

For the multivariate subset selection, I used a CFS method developed by "Advancing Feature Selection Research - ASU Feature Selection Repository". This function uses a correlation based heuristic to evaluate the worth of features which is called CFS [4].

After applying this function to the dataset, I obtain the following subset: ct_defuse_kits', 'bomb_planted', 'time_left', 't_helmets', 'ct_helmets', 'ct_weapon_ak47', 'ct_weapon_awp','ct_players_alive','t_armor','t_weapon_ak47', 'ct_armor', 't_weapon_awp', 'round_winner'.

If we compare this subset with the one obtained in the previous section with the chi-squared function, we see that they have 5 features in common (other 5 non common).

After training the models with this new subset, we obtain their accuracy:

```
Accuracy for k-NN: 0.7274826789838337
Accuracy for RIPPER: 0.6397228637413395
Accuracy for Neural Networks: 0.7192945622506823
Accuracy for Decision Trees: 0.7778710896493807
Accuracy for Svm: 0.7375603611169431
```

***Figure 17**: Accuracy for all the classifiers.*

And plotting the ROC curve:

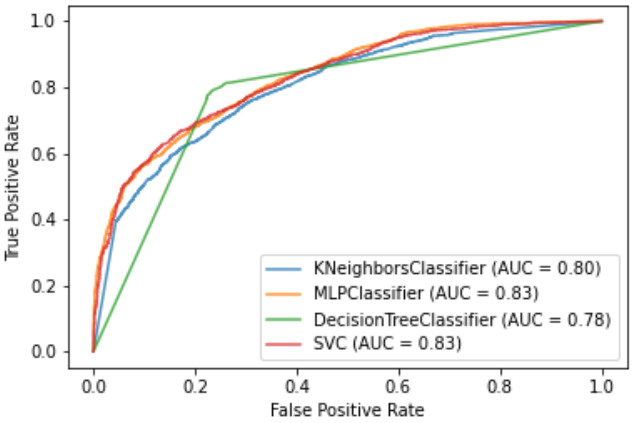

***Figure 18**: ROC curve with multiv. subset selection*

Using this subset, the accuracy of all methods descends slightly too, in exception of the SVM case. With this subset the SVM model improves his accuracy for almost 10%.

In this case, we again lose half of the features (comparing to the original dataset) but models don't lose their predictive power (accuracy descends slightly). In short, we obtain the same result from the univariate filter, with exception of SVM.

To explain what happened with the model, we can remember *Figure 6* and our first analysis, where we concluded that SVM wasn't performing well due to the complexity of the problem. Now that we have simplified our problem reducing the number of features, it seems like the SVM model is making better predictions.

Furthermore, even though we reduced the dataset and the training is therefore faster; if we take into account the time needed to apply the CFS function to our dataset, we have to conclude that applying this multivariate filter to this problem is not worth it. In fact, the accuracy we obtain is really similar to the one obtained in the univariate subset selection method.

IV.IV) WRAPPER SUBSET SELECTION

Finally, we will apply a wrapper filter to our dataset. This function is taken from the mlxtend library, and it is a Sequential Feature Selector. More precisely, it is a Sequential Forward Selecor (SFS).

This wrapper filter is applied individually to each model, so the subset we obtain is different for each one. (I wasn't able to apply this filter to ripper, so this model won't be analyzed this section). The subset we obtain for each model is presented in the following figure, where the numbers (from 0 to 19) are the indexes of the selected columns of the original dataset:

```
[(0, 5, 6, 7, 10, 11, 12, 15, 16, 18),
 (1, 6, 7, 10, 11, 13, 14, 16, 17, 18),
 (3, 5, 10, 11, 13, 14, 15, 16, 17, 18),
 (1, 4, 5, 6, 7, 10, 11, 12, 13, 16)]
```

***Figure 19**: Selected subset for each algorithm (from top to bot):k-NN , NN, Decision Tree, SVM.*

I felt like showing the indexes would be better than showing the 40 names of the features. In this way, we can see how some of the columns are repeated a lot, while others appear in all 4 subsets (16= 't_weapon_ak47', 11='t_helmets', 10='ct_helmets').

It is worth pointing out that obtaining these subsets is extremely time consuming, given that the models have to be trained several times.

The accuracy we obtain after training the models (each one with his own subset):

```
Accuracy for k-NN: 0.7247533067394499
Accuracy for Neural Networks: 0.7474280915389461
Accuracy for Decision Trees: 0.7698929246273357
Accuracy for Svm: 0.7432290573168171
```

***Figure 20**: Accuracy for trained classifiers*

And the correspondent ROC curves are presented on Figure 21.

If we compare the results obtained with this wrapper approach with the ones obtained with the full dataset (all variables), we find out that the accuracy for SVM and NN increases, while the accuracy for k-NN and Descision trees decrease.

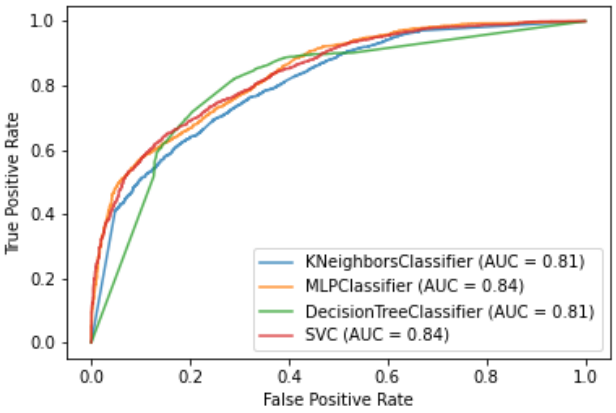

***Figure 21**: ROC curve for Wrapper subset selection*

In the first analysis we concluded that SVM and NN where the simplest classifiers, and that they weren't performing as well as the other models because of the complexity of the problem. Now, we simplified the problem by reducing the number of features, and it seems like this two models are performing much better.

On the other hand, decision trees and k-NN models lose predictive power when the number of features is reduced. Their accuracy is still as good as the others' one, but they were performing better with the whole dataset.

We can conclude that this SFS method is a really useful tool to simplify our problem, and some models that weren't performing properly with a lot of features will increase their predictive power. On contrast, some models that were able to use all the information (with a lot of features), can slightly decrease their accuracy after applying the method.

V. CONCLUSION

First of all, a CS:GO dataset was chosen and after some preliminary analysis we concluded that it was possible to use it for a classification problem. Then, we performed 4 analysis with different subsets of the dataset on 5 different classifiers (svm,NN,k-NN,tree,ripper).

In the first analysis, we saw that there were two models performing better than the others (>80 % accuracy), decision trees and k-NN. Plotting their decision regions (*figure 8 and 10)* helped us to understand that given a dataset with a lot of overlapping points (such as ours), where performing a classification task isn't trivial, this two models were able to find complex patterns in the dataset. We saw how NN and SVM models where building a really simple classifier (*figures 6 and 14*) which is not optimum for our problem.

The rule induction RIPPER algorithm uses rules in the same way decision trees do it, but ripper's rules are not hierarchical so they can't define really complex rules (which are needed in our problem), and that's why this model has not a high accuracy.

In the second analysis, we applied a univariate filter (chi-squared function) to obtain a subset of 10 features, and then we trained new models with that subset. Here we concluded that our models lose accuracy after losing half of the data, but they perform much faster.

For the multivariate subset selection, we used the CFS function, which uses a correlation based heuristic to evaluate the worth of features. This time the accuracy of our models also decreased, in exception of SVM. We concluded that our SVM model was a really simple classifier, so reducing the features and simplifying the problem helps the model perform better.

Finally, we applied a multivariate wrapper selection, obtaining a different subset for each model. After training the models with the new subsets, we realize that simple models (NN, SVM) that had a low accuracy when working with all variables, performed much better with the new subset. Models that were able to find complex patterns and extract information from all the features when working with the whole dataset, lose some accuracy working with this subset. In conclusion, even though we almost lose half of the data (from 20 features to 11), the classifiers are still able to perform decently. In fact, simple models perform even better with this subset. We can undoubtfully state that this wrapper filter can be a really useful tool.

REFERENCES

[1] https://www.destructoid.com/review-counter-strike-global-offensive-233724.phtmlW.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[2] https://www.kaggle.com/christianlillelund/csgo-round-winner-classification

[3] Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[4] Zhao, Zheng et al. "Advancing Feature Selection Research - ASU Feature Selection Repository" 2010.

APPENDIX

In this appendix I located the figures that are too big to fit in the text, so they can be visualized properly.

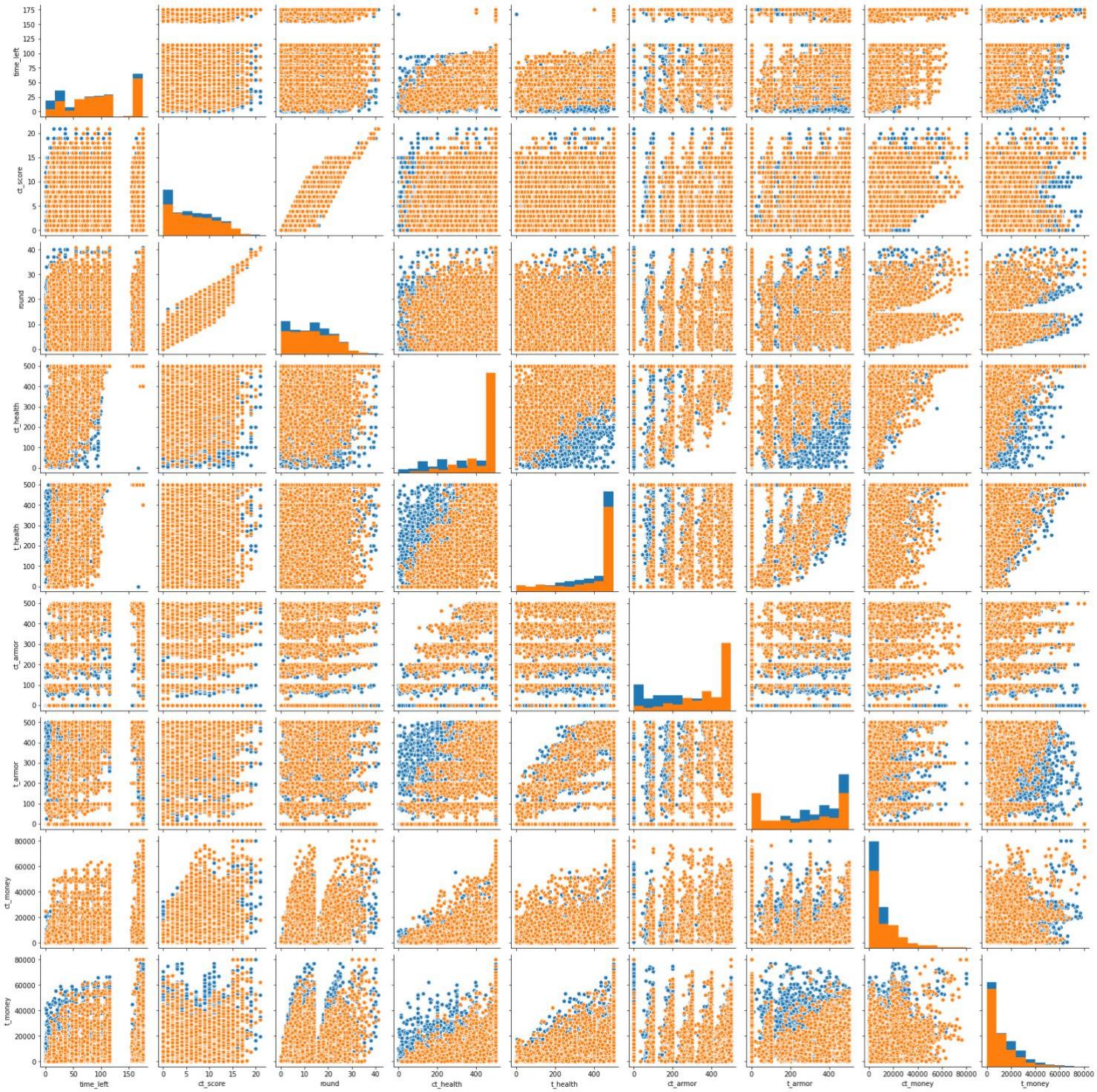

***Figure 1**: All variables (except Boolean or discrete valued ones) of our dataset represented against each other.*

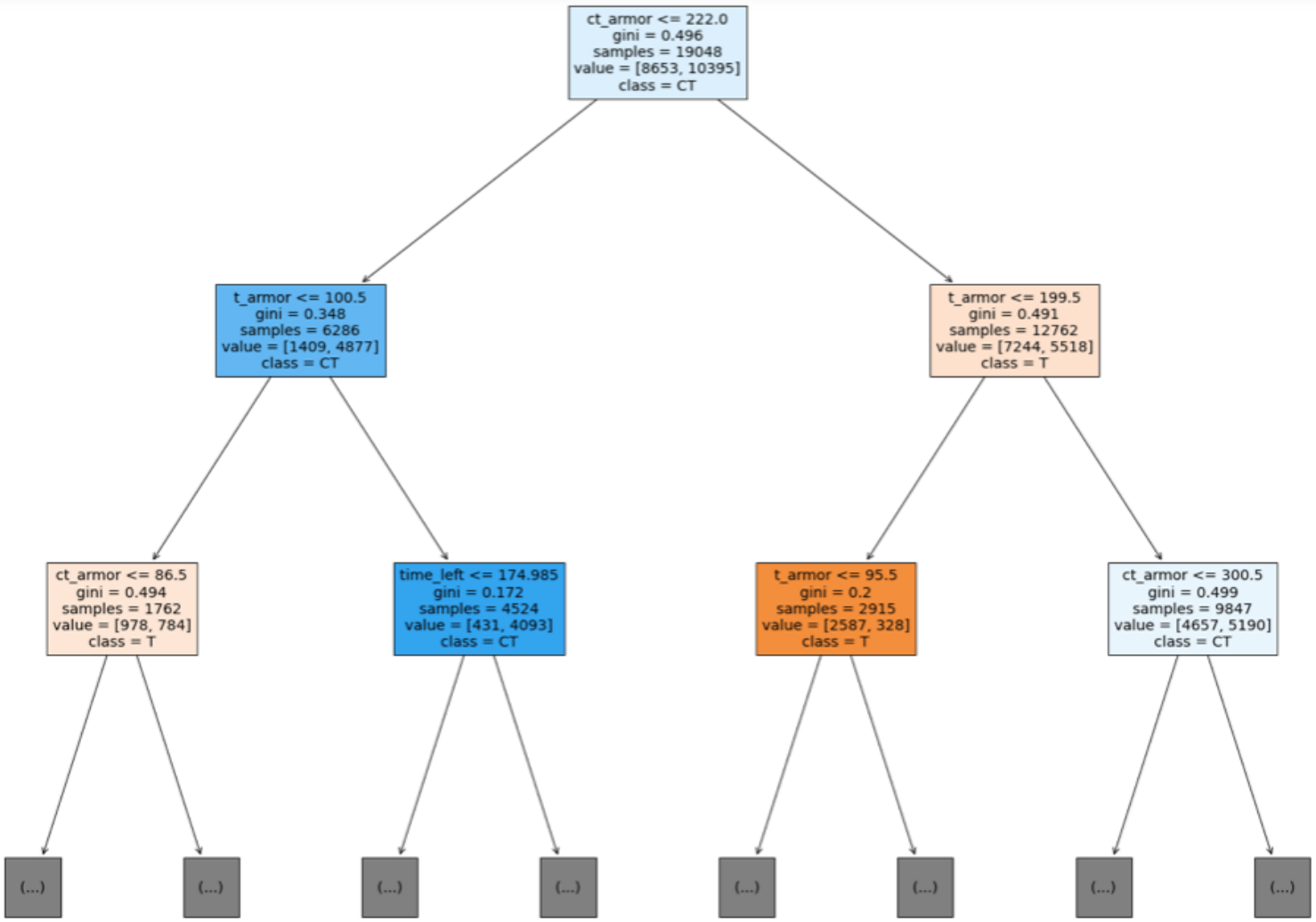

***Figure 11**: Visualization of the first nodes of our trained decision tree.*